เรื่องน่าอ่าน
ศรันย์ ไมตรีเวช

ไมโครซอฟต์ Win32 API

สัมผัสไมโครซอฟต์ Win32 API ที่ช่วยงานให้คุณได้เกินจริง และมีความสามารถหาตัวจับยาก

เป็นเรื่องน่าตื่นเต้นอย่างใหญ่หลวง นับแต่วินโดวส์เวอร์ชัน 3.0 เอ็นฮานซ์โหมดสามารถรับมือกับดีไวซ์ไดรเวอร์ของบัส 32 บิต จะไม่ให้น่าแตกตื่นอย่างไรในเมื่อเคยอ้างตำแหน่งข้อมูลกันอย่างมากถึง 16 เมกะไบต์ และปรับขึ้นพรวดเดียวอ้างได้ถึง 4096 เมกะไบต์ ดู ๆ แล้วก็เหมือนกับคนที่เคยท่องศัพท์ภาษาอังกฤษได้แค่วันละสองคำ จู่ ๆ เจอตะพดวิเศษฟาดหน้าผากโป๊ะเดียว วันต่อมาถึงกับจำได้ทีละตั้ง 130 คำ!

ไมโครซอฟต์ Win32 API ช่วยให้แอพพลิเคชันบนโอเอสตระกูลวินโดวส์ชื่นมื่นไปกับมหิทธิอำนาจแห่งชิพไมโครโปรเซสเซอร์รุ่นใหม่ที่สามารถอ้างหน่วยความจำได้เป็นกิกะไบต์

ทั้งฟังก์ชัน, เมสเสจ, และโครงสร้างล้วนจรรโลงความเป็นน้ำหนึ่งอันเดียวกันระหว่างแพลตฟอร์ม เช่น ชิคาโก วินโดวส์ เอ็นที 3.1 และวินโดวส์ 3.1 ฉะนั้นด้วยการใช้ Win32 API นี้เอง คุณจะพัฒนาโปรแกรมที่รันได้ทุกแพลตฟอร์ม ทั้งนี้โดยไม่เสียประโยชน์จากคุณสมบัติเด่นเฉพาะตัวของแต่ละแพลตฟอร์มด้วย

เมื่อความสามารถเปลี่ยนไปในทางที่ดีขึ้น ก้าวหน้าพัฒนาขึ้นมาขนาดนี้ แน่นอนผลดีย่อมตกอยู่กับผู้ใช้ ใครเคยเล่นหมากรุกคอมพิวเตอร์บนวินโดวส์มาก่อนคงพอจะเข้าใจ มันดูชาญฉลาดขึ้นอย่างผิดหูผิดตา ด้วยเหตุผลก็คือมัน "เห็น" หมากมากกว่าเดิมในเวลาที่เท่ากัน ซึ่งก็สะท้อนความจริงที่ว่าความเร็วมันพุ่งปรู๊ดราวกับรถแข่งติดเทอร์โบ

แล้วโปรแกรมเมอร์อย่างเราท่านล่ะ เอ...จะได้รับผลดีอย่างผู้ใช้หรือเปล่า รีว่าต้องรับกรรม ตกที่นั่งลำบาก ต้องปรับเปลี่ยนเรียนรู้อะไรเพิ่มเติมขึ้นมากมายแค่ไหน?

ทางโน้นเขาก็ปลอบว่าอย่าตีตนไปก่อนไข้เลยเพื่อนเอ๋ย เพื่อนก็น่าเห็นใจอยู่หรอกที่ลงทุนลงแรงไปมากมายกับซอฟต์แวร์บนวินโดวส์ การเปลี่ยนแปลงในทางที่ดีขึ้นนี้เราจะพยายามไม่ทำให้เพื่อนต้องน้ำตาเช็ดหัวเข่าแต่อย่างใด

ความแปลกใหม่ใน Win32 API

เราก็มาดูความจริงกันดีกว่าครับ ผมคิดว่าคุณคงตัดสินใจได้ด้วยตัวเองว่าความเปลี่ยนแปลงในทางดีขึ้นตามที่ท่านว่าไว้นั้น มันก่อความลำบากลำบนอะไรให้คุณหรือเปล่า

ลองมาดูความสามารถของ Win32 API กันว่ามีอะไรบ้าง

● งานจัดการเกี่ยวกับวินโดวส์ (Windows Management)

● อุปกรณ์กราฟิก (GDI Graphics Device Interface)

● บริการจากระบบ (System Services)

● มัลติมีเดีย (Multimedia)

● เรียกโพรซีเยอร์ข้ามแพลตฟอร์ม (Remote Procedure Calls (RPC))

ทางพี่ใหญ่ไมโครซอฟต์แกรับรองครับว่าการทำงานของ Win32 API จะเป็นไปด้วยดีและไม่แตกแถวไปจากกันแม้จะอยู่คนละแพลตฟอร์ม อาจมีข้อยกเว้นบางประการซึ่งก็น้อย ซึ่งเดี๋ยวจะกล่าวถึงเป็นเรื่อง ๆ ไป

ตอนนี้ขอเกริ่นเพียงคร่าว ๆ ว่าความแตกต่างระหว่างการใช้ Win32 API ขึ้นอยู่กับลักษณะเด่นของแต่ละแพลตฟอร์ม ความแตกต่างที่น่าจับตาที่สุดก็คือ Win32 API บางตัวสามารถทำงานได้เฉพาะบนแพลตฟอร์มที่เหนือกว่าในด้านหนึ่ง ๆ เป็นต้นว่าฟังก์ชันเกี่ยวกับความปลอดภัยข้อมูลจะมีก็แต่บนวินโดวส์ เอ็นทีเท่านั้น ความแตกต่างอื่น ๆ โดยมากจะเป็นเรื่องข้อจำกัดของระบบ อย่างเช่นขีดจำกัดของช่วงค่าข้อมูล หรือจำนวนรายการที่ฟังก์ชันหนึ่ง ๆ สามารถรับมือได้

ในส่วนของข้อจำกัดของระบบเราจะมาดูกันอีกทีในหัวข้อเฉพาะของมัน ตอนนี้เรามาไล่ดูกันอย่างละเอียดทีละข้อ ว่าด้วยหลักการทำงานของ Win32 API

งานจัดการเกี่ยวกับวินโดวส์

ทำให้แอพพลิเคชันของคุณสามารถสร้างและจัดการเกี่ยวกับเรื่องติดต่อกับผู้ใช้ตามสไตล์วินโดวส์ เป็นต้นว่าสร้างหน้าต่างเพื่อแสดงอินพุต/เอาต์พุต และใช้ฟังก์ชันในส่วนที่เกี่ยวกับการบริหารวินโดวส์สร้างขึ้นมาติดต่อกับผู้ใช้ได้เลย

แอพพลิเคชันจะบ่งบอกพฤติกรรมและลักษณะการปรากฏของตัวเองโดยสร้างวินโดวส์คลาส และตอบสนองโพรซีเยอร์ของวินโดวส์ โดยที่วินโดวส์คลาสจะบ่งบอกลักษณะสามัญ เป็นต้นว่าวินโดวส์นั้น ๆ มีเมนูหรือเปล่า ในวินโดวส์จะมีโพรซีเยอร์ซึ่งกำหนดพฤติกรรมของตัวมันเอง, การตอบสนองต่อผู้ใช้ เช่นดับเบิ้ลคลิ๊กแล้วจะมีปฏิกิริยาอย่างไร, และประมวลผลข้อมูลที่ผู้ใช้คีย์เข้ามา

ถ้าใครสงสัยว่าเมื่อมีหน้าต่างหลายบานแสดงพร้อมกันบนหน้าจอวินโดวส์มีอะไรมาแยกแยะว่าผู้ใช้คลิ๊กส่วนนั้นส่วนนี้ของหน้าจอแล้วควรจะไปเกิดเหตุการณ์กับแอพพลิเคชันตัวใด คำตอบก็คือแอพพลิเคชันแต่ละตัวจะไม่รับการติดต่อจากผู้ใช้ (เช่น คลิ๊กเม้าส์) จากอาณาบริเวณทั้งหมดบนหน้าจอ แต่ระบบของวินโดวส์จะแบ่งสรรปันส่วนพรมแดนที่แน่นอนยิ่ง (วัดกันเป็นพิกเซล) ให้กับหน้าต่างที่ได้รับการติดต่อหมายถึงถ้าเม้าส์พอยน์เตอร์มาลงภายในอาณาบริเวณเท่านั้นเท่านี้ก็จะมีหน้าต่างแอพพลิเคชันที่ครอบครองอยู่ตอบรับการติดต่อจากผู้ใช้ไป กระบวนการเหล่านี้ใช้ฟังก์ชันประเภท GDI (Graphics Device Interface) เมื่อขนาดหรือตำแหน่งของหน้าต่างแอพพลิเคชันเปลี่ยนไป ระบบจะส่งเมสเสจไปยังแอพพลิเคชันเพื่อสั่งให้ทำการเพนต์ส่วนของวินโดวส์ที่มันบังไว้ (คือเมื่อหน้าต่างแอพพลิเคชันโผล่ขึ้นทับอะไรไว้ มันจะเป็นผู้รู้และเก็บรายละเอียดนั้น ๆ ไว้ เมื่อถูกสั่งให้ลดขนาดก็จึงต้องเป็นผู้รับภาระในการเพนต์ส่วนที่ตนบังเอาไว้ขึ้นมาใหม่)

แอพพลิเคชันรับสัญญาณอินพุตจากเม้าส์และคีย์บอร์ดในรูปของเมสเสจ โดยที่ระบบจะเป็นผู้ "ผู้แปล" ความเคลื่อนไหวของเม้าส์, การกดปุ่ม, หรือการพรมนิ้วลงบนคีย์บอร์ด ไปเป็นอินพุตเมสเสจ และวางเมสเสจเหล่านี้ลงบนคิว ในการนี้จึงขอกล่าวเสียเลยว่าระบบของวินโดวส์นะครับที่เป็นผู้จัดการเกี่ยวกับเมสเสจคิว ไม่ใช่ตัวแอพพลิเคชัน เพราะถ้าขืนให้พวกมันจัดคิวกันเองมีหวังตีกันยุ่งล่ะครับ

แต่แอพพลิเคชันเองจะเป็นผู้ใช้เมสเสจฟังก์ชันในการดึงเอาเมสเสจออกมาจากคิวและส่งด่วนไปให้กับโพรซีเยอร์ที่เหมาะสมเพื่อทำการประมวลผล

อันที่จริงแอพพลิเคชันสามารถประมวลผลสัญญาณจากเม้าส์และคีย์บอร์ดได้โดยตรง หรือไม่ก็ให้ระบบเป็นผู้แปลสัญญาณอินพุตระดับต่ำเหล่านี้เป็นคอมมานด์เมสเสจด้วยการใช้เมนูและฮอตคีย์ (เป็นต้นว่ากดปุ่ม Alt+F4 เพื่อจบการทำงานแอพพลิเคชัน) การที่คุณใช้เมนูนั้นก็เพื่อตั้งใจที่จะลิสต์คำสั่งที่เป็นไปได้ทั้งหมดให้ผู้ใช้ ระบบจะจัดการให้ผู้ใช้เลือกคำสั่ง ไม่ว่าจะเลือกด้วยเม้าส์หรือคีย์บอร์ดก็ตาม แล้วส่งเมสเสจที่บ่งบอกถึงตัวเลือกไปยังโพรซีเยอร์อันสมควร

แอพพลิเคชันมักทำตัวเป็นพี่เลี้ยงให้แก่ผู้ใช้ อาจจะทั้งก่อนและหลังอินพุต การให้ความช่วยเหลือจะมาในรูปของไดอะล็อกบ็อกซ์ ซึ่งเป็นหน้าต่างชั่วคราวที่ปรากฏขึ้นเพื่อแสดงข้อมูลหรือเรียกร้องอินพุต เป็นต้นว่าให้คลิ๊กเลือกปุ่ม Yes หรือ No อย่างใดอย่างหนึ่ง ไดอะล็อกบ็อกซ์จะเป็นผู้จัดการและประมวลผลอินพุตของผู้ใช้ที่มีเข้ามาด้วยตัวเอง มันเป็นตัวช่วยให้การ "สั่ง" จากผู้ใช้

สัมฤทธิ์ผลตามความตั้งใจ

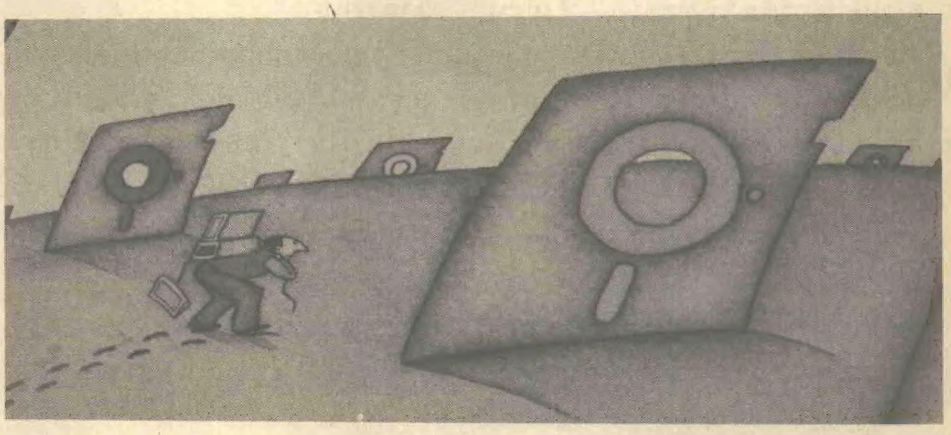
นอกจากนี้ฟังก์ชันบริหารวินโดวส์ยังให้ฟีเจอร์อื่น ๆ ที่เกี่ยวข้องกับวินโดวส์อีก อย่างเช่นฟังก์ชันเกี่ยวกับคลิปบอร์ดให้สิทธิ์ในการ copy/paste ภายในวินโดวส์เดียวกันหรือข้ามวินโดวส์ทั้งที่ต่างและเป็นแอพพลิเคชันเดียวกัน แอพพลิเคชันยังใช้ฟังก์ชันเกี่ยวกับคลิปบอร์ดในการปฏิบัติงาน DDE (Dynamic Data Exchange) โดยที่การปฏิบัติงาน DDE จะให้แอพพลิเคชันทำการแลกเปลี่ยนข้อมูลได้โดยไม่จำเป็นต้องอาศัยผู้ใช้มาเป็นผู้ชี้นำแต่อย่างใด

GDI (Graphics Device Interface)

GDI ให้ฟังก์ชันและโครงสร้างที่เกี่ยวข้องซึ่งแอพพลิเคชันของคุณสามารถใช้เพื่อที่จะสร้างการให้ผลทางกราฟิก นับแต่การแสดงบนจอ การพิมพ์ด้วยพรินเตอร์ และอุปกรณ์ที่เกี่ยวกับกราฟิกอื่น ๆ ด้วยเหล่าฟังก์ชัน GDI นี้ คุณสามารถวาดเส้นสายลายโค้ง ทรงเหลี่ยมทรงกลม อักษรต่าง ๆ ตลอดไปจนกระทั่งภาพบิตแมพ ส่วนเรื่องสีสันหรือสไตล์ของการเกิดรูปขึ้นอยู่กับออปเจ็กต์ที่คุณสร้างขึ้นมาใช้วาดว่าจะเป็นปากกา (pen), แปรง (brush), และแบบตัวอักขระ (font)

อย่างเช่นถ้าต้องการวาดเส้นตรงเส้นโค้งก็ใช้ปากกา ถ้าจะระบายสีก็ใช้แปรง และถ้าจะให้เป็นตัวอักษรหรือตัวเลขก็ใช้แบบตัวอักขระ

แอพพลิเคชันเป็นผู้กำหนดส่งเอาต์พุตไปยังอุปกรณ์หนึ่ง ๆ โดยสร้างดีไวซ์คอนเท็กซ์สำหรับอุปกรณ์ดังกล่าว ดีไวซ์คอนเท็กซ์เป็นโครงสร้างที่ GDI



ใช้เก็บรายละเอียดต่าง ๆ ของอุปกรณ์ เป็นต้นว่ามันทำงานที่โหมดไหน ตัวเลือกต่าง ๆ ในปัจจุบันเป็นอะไร แอพพลิเคชันหนึ่ง ๆ จะสร้างดีไวซ์คอนเท็กซ์ด้วยฟังก์ชันที่อุทิศให้เรื่องของดีไวซ์คอนเท็กซ์โดยเฉพาะ

GDI จะส่งแฮนเดิ้ลของดีไวซ์คอนเท็กซ์ที่จะใช้ในการเรียกครั้งต่อไปเพื่อกำหนดอุปกรณ์ เป็นต้นว่าแอพพลิเคชันต้องการดูข้อมูลเกี่ยวกับขอบเขตความสามารถของอุปกรณ์ ความละเอียดของการแสดงผล

แอพพลิเคชันสามารถคุมการส่งเอาต์พุตไปยังอุปกรณ์จริง ๆ อย่างเช่นพรินเตอร์ หรือจะส่งไปยังอุปกรณ์เทียมอันเป็นนามธรรมที่เอาไว้หลอกคอมพิวเตอร์ฝ่ายเดียว อย่างที่เรียกกันว่าลอจิคอลดีไวซ์นั่นแหละครับ ยกตัวอย่างได้แก่หน่วยความจำซึ่งทำตัวเป็นผู้รับเอาต์พุตมาเก็บไว้ในรูปแบบที่ง่ายต่อการส่งไปยังอุปกรณ์จริง ๆ เป็นต้น

แอพพลิเคชันจะใช้กลุ่มฟังก์ชันแอตทริบิวต์ในการเซตโหมดปฏิบัติการ (operating mode) และค่าตัวเลือกปัจจุบันของอุปกรณ์โหมดปฏิบัติการนั้นหมายเอาทั้งสีของอักขระและพื้นหลัง, การผสมสี, ออปเจ็กต์ที่ใช้ในการวาด (เช่นปากกาหรือแปรง)

การบริการจากระบบ

การบริการจากระบบมาในรูปของเซตฟังก์ชันที่ช่วยให้แอพพลิเคชันสามารถใช้ทรัพยากรระบบและสมบัติต่าง ๆ ของโอเอส เป็นต้นว่าหน่วยความจำ ระบบไฟล์ และการประมวลผล แอพพลิเคชันหนึ่ง ๆ ใช้ฟังก์ชันกลุ่มนี้เพื่อบริหารและติดตามดูความเป็นไปของทรัพยากรที่แอพพลิเคชันต้องใช้ อย่างเช่นใช้ฟังก์ชันเกี่ยวกับการบริหารหน่วยความจำเพื่อหาที่ลงให้กับข้อมูล และใช้ฟังก์ชันเกี่ยวกับการบริหารงานประมวลผลเพื่อติดต่อประสานงานกับแอพพลิเคชันอื่น

การบริการจากระบบช่วยให้แอพพลิเคชันของคุณเข้าไปสู่ไฟล์, ไดเรคทอรี, และอุปกรณ์อินพุตเอาต์พุตทั้งหลายได้ เช่น ฟังก์ชัน file I/O ช่วยให้แอพพลิเคชันเข้าถึงไฟล์และไดเรคทอรีบนดิสก์และอุปกรณ์เก็บข้อมูลอื่น ๆ ในเน็ตเวอร์ก ฟังก์ชันเหล่านี้สนับสนุนระบบไฟล์หลายชนิด นับตั้งแต่ FAT ของเอ็มเอสดอสไปจนถึง NTFS (Windows NT File System)

เน็ตเวอร์กฟังก์ชันเป็นผู้สร้างและบริหารการติดต่อเพื่อร่วมใช้ทรัพยากรอย่างเช่นไดเรคทอรีและพรินเตอร์ใน

คอมพิวเตอร์ของเน็ตเวอร์ก ฟังก์ชันเกี่ยวกับการสื่อสารจะเป็นผู้อ่านและเขียนจากพอร์ตการสื่อสาร รวมทั้งควบคุมโหมดปฏิบัติการของพอร์ตเหล่านี้

ฟังก์ชันบริการจากระบบยังช่วยให้การเข้าถึงข้อมูลระบบและแอพพลิเคชันอื่นเป็นไปได้ โดยมีฟังก์ชันเฉพาะเกี่ยวกับข้อมูลระบบซึ่งจะช่วยบอกให้แอพพลิเคชันให้ทราบถึงสเปกของคอมพิวเตอร์ เป็นต้นว่าเม้าส์กำลังแอคทีฟอยู่หรือเปล่า จอภาพเป็นอย่างไร และจะมีฟังก์ชันอยู่พวกหนึ่งที่ทำหน้าที่เก็บข้อมูลเกี่ยวกับตัวแอพพลิเคชันเองลงในไฟล์ระบบ เพื่อที่แอพพลิเคชันอื่นจะสามารถเรียกขึ้นมาดูได้

การบริการจากระบบช่วยให้แอพพลิเคชันสามารถร่วมใช้ข้อมูลกับแอพพลิเคชันที่รันอยู่ในเครื่องเดียวกันหรือเครื่องอื่นภายในวงเน็ตเวอร์ก แอพพลิเคชันสามารถคัดลอกข้อมูลในระหว่างการประมวลผลโดยใช้ mailslot กับ pipe ฟังก์ชัน เพื่อประกอบกิจการสื่อสารแบบประมวลผลร่วม (interprocess communication - IPC)

สำหรับระบบปฏิบัติการ (OS) ที่มีฟีเจอร์เกี่ยวกับความปลอดภัยในการเข้าถึงข้อมูล จะมีฟังก์ชันเกี่ยวกับ security ที่จะทำให้แอพพลิเคชันมีทางเข้าได้ถึงข้อมูลที่ถูกปกป้องเหล่านั้น รวมทั้งทำให้แอพพลิเคชันมีส่วนในการช่วยระงับการเข้าถึงข้อมูลหรือความเสียหายบางอย่างด้วย

แอพพลิเคชันของคุณสามารถใช้สมบัติของฟังก์ชันบริการจากระบบเพื่อจัดการกับเงื่อนไขพิเศษระหว่างรัน เป็นต้นว่าจะจัดการกับข้อผิดพลาดที่เกิดขึ้นแต่ละอย่างเช่นใด รวมทั้งรับมือกับข้อยกเว้นบางประการที่อาจถูกกำหนดไว้ล่วงหน้า

นอกจากนั้นยังมีฟีเจอร์ที่แอพพลิเคชันสามารถใช้เพื่อดีบักและปรับปรุงการทำงาน เช่นฟังก์ชันในการดีบักซึ่งอนุญาตให้คุณเอ็กซีคิวต์ทีละสเตป หรือฟังก์ชันดูผลการทำงานที่กำหนด path ตลอดสายการเอ็กซีคิวต์

มัลติมีเดีย

ฟังก์ชันที่เกี่ยวกับมัลติมีเดียช่วยให้แอพพลิเคชันของคุณติดต่อกับเครื่องเสียงและเครื่องแสดงภาพได้ คุณจะรู้สึกว่าแอพพลิเคชันของคุณยิ่งใหญ่ขึ้นมาอย่างไม่น่าเชื่อ อาจเผลอคิดไปได้ว่าโอ้โฮนี่ฝีมือเราหรือ? (จริง ๆ แล้วไม่ใช่หรอกครับ ฝีมือ Win32 API ต่างหาก) ผู้ใช้แอพพลิเคชันของคุณจะมีวิธีสื่อสารกับคอมพิวเตอร์ด้วยรูปแบบที่ไม่ซ้ำกับผู้ใช้รุ่นพี่

นอกจากสารัตถะในการสื่อสารอันน่าตื่นหูตื่นตาแล้ว ฟังก์ชันเกี่ยวกับมัลติมีเดียยังสามารถสร้างเอกสารหรือรูปแบบการเสนองาน เช่น ขึ้นจอสกรีนขนาดใหญ่ มีดนตรีและซาวนด์เอฟเฟคท์ประกอบการเสนองานของคุณอาจเป็นกราฟิกหรืออิมเมจที่ยิ่งใหญ่ตระการตาราวกับภาพยนตร์ฮอลลีวู้ด

โดยรวมแล้วฟังก์ชันเกี่ยวกับมัลติมีเดียให้บริการทั้งกับเครื่องแสดงภาพและเสียง, ไฟล์ไอโอ, เครื่องมือควบคุมอุปกรณ์, จอยสติ๊ก, และไทม์เมอร์

แอพพลิเคชันจะใช้ฟังก์ชันเกี่ยวกับเสียงเพื่อเล่นและเก็บข้อมูลโดยใช้รูปสัญญาณคลื่นหรือดิจิตอล (MIDI) โดยจะมีสิ่งที่เรียกว่า Audio Compression Manager เป็นตัวควบคุมคุณภาพการจัดเก็บข้อมูลเกี่ยวกับเสียง

แอพพลิเคชันจะใช้ฟังก์ชันเกี่ยวกับภาพเพื่อจับวีดีโอคลิป (video clip หมายถึงกลุ่มภาพต่อเนื่องที่นำมาต้มตุ๋นสายตาบรรดาเราท่านว่าเป็นภาพเดียวเคลื่อนไหวได้นั่นแหละครับ), ทำคลิปอัดแน่น, และควบคุมการแสดงภาพ

แอพพลิเคชันจะจับกลุ่มภาพมาจากกล้องถ่ายภาพทั่วไปด้วยเมสเสจธรรมดา ๆ นี่แหละ ขอให้มีฮาร์ดแวร์อันเป็นเครื่องต่อที่ถูกต้องเท่านั้น เราเรียกเครื่องต่อดังกล่าวว่า MCI (Media Control Interface)

เครื่องควบคุมคุณภาพการจัดเก็บข้อมูลเกี่ยวกับภาพก็ชื่อคล้ายเครื่องมือควบคุมคุณภาพเสียง (ทั้งนี้เพราะมันทำงานโดยใช้หลักการเดียวกันคือคอมเพรสและดีคอมเพรส) นั่นคือ Installable Compression Manager ข้อมูลเกี่ยวกับภาพที่เก็บไว้จะสามารถนำมาฉายซ้ำได้ทั้งบนจอคอมพิวเตอร์และอุปกรณ์อื่นซึ่งก็สามารถทำได้ด้วยการติดต่อผ่าน MCI ตามเคย การติดต่อผ่าน MCI นี้แอพพลิเคชันจะเรียกใช้ฟังก์ชันพิเศษซึ่งเป็นวินโดวส์คลาสชื่อ MCIWND

แอพพลิเคชันใช้ฟังก์ชันเกี่ยวกับไฟล์ไอโอเพื่อเก็บและดึงข้อมูลทางมัลติมีเดียต่างชนิดกัน แอพพลิเคชันสามารถใช้ไอโอทั้งที่มีและไม่มีบัฟเฟอร์กับมัลติมีเดียไฟล์, เข้าถึง RIFF ไฟล์, และประสานเหล่าฟังก์ชันคัสตอมไอโอสำหรับชนิดข้อมูลของมัลติมีเดียให้เป็นอันหนึ่งอันเดียวกัน

สิ่งสำคัญที่น่าพูดถึงคือ ไฟล์ฟอร์แมตซึ่งเก็บสัญญาณภาพร่วมกับเสียงเป็นดิจิตอล นั่นคือ AVI (Audio-Video Interleaved) ไฟล์ดังกล่าวเป็น RIFF ไฟล์ชนิดหนึ่งซึ่งสามารถดัดแปลงให้จัดเก็บกลุ่มข้อมูลที่ต่างจากมาตรฐานได้

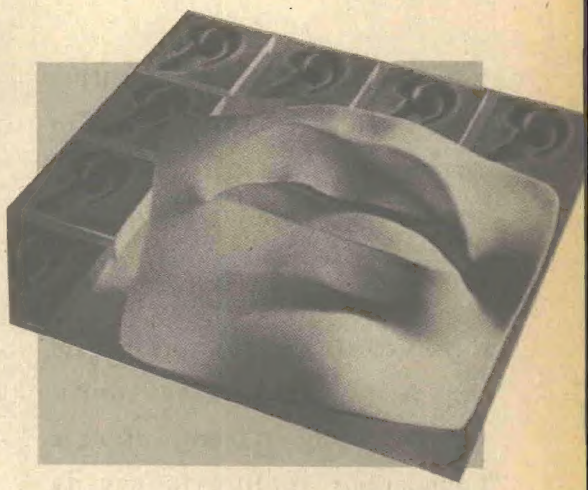
MCI ให้เซตคำสั่งสามัญที่แอพพลิเคชันสามารถนำไปใช้ควบคุมอุปกรณ์ต่าง ๆ เช่น อุปกรณ์เกี่ยวกับแอนนิเมชั่น, MIDI ซีเควนเซอร์, อุปกรณ์ตัดต่อภาพเสียง, วีดีโอดิสค์, วิสก้าเทปเรคอร์ดเดอร์ (VCR), และอุปกรณ์ทางเสียงที่เป็นดิจิตอล เพื่อที่จะติดต่อกับอุปกรณ์ต่าง ๆ แอพพลิเคชันจะส่งเมสเสจ หรือคำสั่งผ่าน MCI แล้ว MCI จะแปลสัญญาณที่มาจากเมสเสจจากนั้นจึงทำการควบคุมให้อุปกรณ์อันเป็นเป้าหมายให้ทำงานตรงตามคำสั่งต่อไป

แอพพลิเคชันใช้ฟังก์ชันเกี่ยวกับจอยสติ๊กเผื่อไว้ให้คุณใช้จอยสติ๊กได้สองอันพร้อมกัน โดยที่แอพพลิเคชันสามารถดึงเอาข้อมูลเกี่ยวกับจอยสติ๊กเปรียบเทียบน้ำหนักได้ด้วย (เช่นที่คุณออกแรงกดไม่เท่ากันแล้วให้ผลลัพธ์ต่างกันในจอยสติ๊กบางรุ่น) และรับเมสเสจเกี่ยวกับการเคลื่อนไหว (ของเม้าส์/จอยสติ๊ก) กับการกด/คลิ๊กปุ่ม

RPC (Remote Procedure Calls)

RPC ทำให้แอพพลิเคชันสามารถปฏิบัติการคำนวณแบบกระจาย (Distributed computing) ซึ่งหมายความว่าแอพพลิเคชันอาจใช้ทรัพยากรและหยิบยืมความสามารถทางการคำนวณจากคอมพิวเตอร์บนเน็ตเวอร์ก

การใช้ RPC ทำให้คุณสามารถสร้างแอพพลิเคชันแบบกระจาย ซึ่งแต่ละหน่วยจะประกอบไปด้วยคอมพิวเตอร์ "ผู้เรียกใช้" (Client) หนึ่งตัวที่ทำหน้าที่แสดงข้อมูลแก่ผู้ใช้ และคอมพิวเตอร์ "ผู้บริการ" (Server) อีกหนึ่งตัวที่ทำหน้าที่เก็บ, ดึง, และดำเนินการกับข้อมูลกับทั้งรับมือการคำนวณ เพื่อ "ผู้เรียก" ตัวอย่างของแอพพลิเคชันแบบกระจายได้แก่การแชร์ดาต้าเบส, ไฟล์เซอร์ฟเวอร์จากคอมพิวเตอร์อื่น, และพรินเตอร์เซอร์ฟเวอร์จากคอมพิวเตอร์อื่น



แอพพลิเคชันแบบกระจายจะรันเสมือนเป็นการประมวลผลในพื้นที่หนึ่งแอดเดรส โดยที่จริง ๆ แล้วตัวมันจะเป็นผู้เรียกโพรซีเยอร์ซึ่งเอ็กซีคิวต์อยู่ในพื้นที่หนึ่งแอดเดรสของคอมพิวเตอร์อีกตัว

สำหรับแอพพลิเคชันแล้วการเรียกดังกล่าวดูเหมือนจะเป็นการกระทำ "ภายใน" อันเป็นปกติธรรมดา แต่อันที่จริงมันไปเกี่ยวข้องกับโพรซีเยอร์ส่วนหนึ่งของไลบรารีของ RPC โดยที่ RPC จะบริหารการสื่อสารเน็ตเวอร์กให้

การแยกแยะโครงสร้างเครือข่าย

การแยกแยะโครงสร้างเครือข่ายนั้น ทำกันตามลักษณะผู้ใช้งาน สามารถเรียกได้ดังนี้

1. การคำนวณแบบกลุ่ม (Workgroup) ใช้ทรัพยากรร่วมกัน เช่น ฮาร์ดดิสค์, พรินเตอร์ตัวเดียวกัน ทำงานเดียวกัน แต่จุดสำคัญคือทำคนละส่วนและสามารถแบ่งข้อมูลหรือโปรแกรมระหว่างกัน ทำให้เสียค่าใช้จ่ายน้อยลงในขณะที่ประสิทธิภาพเพิ่มขึ้น

2. การคำนวณแบบกระจาย (Distributed) ใช้ทรัพยากรร่วมกัน ใช้ความสามารถทางการคำนวณอันทรงประสิทธิภาพของคอมพิวเตอร์เครื่องหนึ่งหรือหลายเครื่องบนเน็ตเวอร์ก จุดสำคัญคือต่างคนต่างรักษาฐานข้อมูลส่วนของตัวเองไว้ โดยมีจุดมุ่งหมายในการนำมารวมกันให้เป็นระบบหนึ่งเดียว พูดง่าย ๆ ข้อมูลทั้งหมดไม่นำมาตั้งไว้ในที่เดียวกัน แต่เวลาใช้จะเห็นข้อมูลได้ทั่วถึง ทำแบบนี้มีผลให้มีพื้นที่เก็บข้อมูลมหาศาล การเรียกข้อมูลก็ไม่ชักช้าด้วย

3. การคำนวณแบบเครือข่าย (Network) ใช้ทรัพยากรแยกกัน แต่ทำงานในระบบเดียวกัน จุดสำคัญคือข้อมูลอยู่ที่เดียวกัน ซึ่งอันนี้ก็เป็นคอนเซ็ปต์ค่อนข้างโบราณตั้งแต่ยังใช้เมนเฟรมกันอยู่

4. การคำนวณแบบไคล์เอ็นต์เซอร์ฟเวอร์ (Client/Server) เป็นรูปแบบการใช้ทรัพยากรที่ทันสมัยมาก นั่นคือแต่ละเครื่องอาจขอใช้ทรัพยากรของอีกเครื่องหนึ่งผ่านเครือข่าย ทำให้มีการแบ่งสันปันส่วนกันได้ประโยชน์สูงสุด การประยุกต์ใช้งานเป็นไปได้หลากหลายในขณะที่การลงทุนต่ำลงทั้งฮาร์ดแวร์และซอฟต์แวร์ แถมยังเป็นระบบเปิด ไม่ขึ้นกับค่ายไหนยี่ห้อใดอีกด้วย

ซึ่งสนับสนุนการเรียกดังกล่าว RPC มีเอี่ยวไปกระทั่งรายละเอียดอย่างโพรโทคอลของเน็ตเวอร์ก ซึ่งนี่ก็แปลได้ว่าแอพพลิเคชันแบบกระจายแทบไม่ต้องการโค้ดเกี่ยวกับเน็ตเวอร์กเลย

RPC ของไมโครซอฟต์เป็นเพียงส่วนหนึ่งของสภาพแวดล้อมที่สมบูรณ์เพื่อการคำนวณแบบกระจายซึ่งถูกกำหนดโดย OSF (Open Software Foundation) อันเป็นการตกลงธุรกิจสากลระหว่างบริษัท เพื่อทำการกำหนดนิยามองค์ประกอบของสภาพแวดล้อมที่สมบูรณ์ซึ่งสนับสนุนการคำนวณแบบกระจาย

RPC ของไมโครซอฟต์เข้าใช้กันได้เกือบสนิทกับมาตรฐานของ OSF (สังเกตคำว่า "เกือบสนิท" ดี ๆ นะครับ มันแปลว่าไม่สนิทเต็มร้อยนั่นแหละ เขาว่าเป็นแค่เรื่องปลีกย่อยที่ไม่สลักสำคัญอะไร)

แอพพลิเคชันของผู้เรียกหรือผู้บริการที่เขียนขึ้นโดยใช้ MS RPC เวอร์ชัน 1.0 จะเทียบค่าเอากับ DCE RPC ของผู้เรียกหรือผู้บริการซึ่งรันไทม์ไลบรารี (run-time libraries) ติดตั้งฐานการติดต่อและรันบนโพรโทคอลที่สนับสนุน

Extension Libraries

ไลบรารีเสริมจะเพิ่มความสามารถและให้บริการแอพพลิเคชันของคุณเกินกว่าการบริการพื้นฐานของ Win32 API นอกจากมันจะขยายขอบเขตการบริการที่มีอยู่แล้วใน Win32 API มันยังให้บริการเฉพาะกิจ ซึ่งใช้กันเป็นปกติในแอพพลิเคชันบน Win32 เพิ่มเติมอีกด้วย และต่อไปนี้คือไลบรารีเสริมเหล่านั้น

1. Common Control
2. Common Dialog Boxes
3. Data Decompression
4. File Installation
5. DDE Management (Dynamic Data Exchange Management)
6. Network Dynamic Data Exchange
7. Shell

ทั้งไลบรารี Common Control และ Common Dialog Boxes ต่างก็ทำให้งานสร้างและบริหารคอนโทรลกับไดอะล็อกบ็อกซ์ง่ายขึ้น โดยที่ไลบรารี Common Control จะเตรียมคอนโทรลหลัก ๆ ไว้ให้สำเร็จรูป มีจุดมุ่งหมายให้ผู้ใช้ไม่จำเจกับการติดต่อสื่อสารและรับรู้ข้อมูลในรูปแบบใดรูปแบบหนึ่งมากเกินไป ส่วนไลบรารี่ Common Dialog Boxes นั้นจะเตรียมไดอะล็อกบ็อกซ์สำเร็จรูปไว้ให้ โดยมีจุดประสงค์ในการใช้งานถามความแน่ใจหรือต้องการอินพุตบางขั้นตอน เช่น เปิดปิดไฟล์ เซฟข้อมูล หรือพิมพ์ข้อมูลอะไร

ส่วนไลบรารี Data Decompression และ File Installation นั้นจัดเตรียมฟังก์ชันที่เป็นประโยชน์ต่อแอพพลิเคชันที่มีการอินสตอลไฟล์ ไลบรารี Data Decompression ชื่อก็บอกอยู่แล้วว่ามีหน้าที่ขยายไฟล์ที่ถูกอัดแน่น (Compressed file) โดยไฟล์ยูทิลิตี้ของไมโครซอฟต์ที่ชื่อ COMPRESS.EXE ไลบรารีนี้ เมื่อแอพพลิเคชันสามารถใช้ได้ก็หมายความว่าคุณสามารถสั่งขยายไฟล์อัดแน่นจากแอพพลิเคชันของคุณได้นั่นเอง ส่วนไลบรารี File Installation นั้นจะเตรียมฟังก์ชันที่ทำให้แอพพลิคชันของคุณสามารถวิเคราะห์ไฟล์ที่ถูกอินสตอลไปแล้วและอินสตอลไฟล์ใหม่เสริมเข้าไปได้

ไลบรารี DDE Management และ Network DDE ทำให้งานประมวลผลการแลกเปลี่ยนข้อมูลข้ามแอพพลิเคชันเป็นไปได้ง่ายขึ้น โดยที่ไลบรารี DDE Management จัดเตรียมฟังก์ชันที่ช่วยลดโค้ดในแอพพลิเคชันของคุณให้เหลือน้อยที่สุดเท่าที่จะเป็นไปได้ นอกจากนี้ยังทำให้ผู้ใช้นั่งเฉย ๆ เป็นตอไม้ระหว่างงาน DDE เกิดขึ้น ส่วน Network DDE นั้นจัดเตรียมฟังก์ชันที่แอพพลิเคชันใช้ในการติดต่อกับ DDE เซอร์ฟเวอร์บนคอมพิวเตอร์เครื่องอื่น ๆ ในระบบเครือข่าย ฟังก์ชันเหล่านี้จะช่วยลดโค้ดและเพิ่มความแน่นอนในการส่งข้อมูลข้ามเครื่องอีกด้วย

สรุปแล้วมีอะไรใหม่บ้างหรือเปล่า?

เฉพาะรีลีสนี้ของ Win32 API เราได้ฟังก์ชันและเมสเสจใหม่ ๆ เพิ่มขึ้นมาก ซึ่งก็ทำให้แอพพลิเคชันสามารถเข้าถึงฟีเจอร์ 32 บิตของระบบปฏิบัติการชิคาโกในขอบเขตของการบริหารวินโดวส์, GDI, บริการจากระบบ, และไลบรารีเสริม

ข้อจำกัดของชิคาโก

มีฟังก์ชันและเมสเสจหลายตัวที่ต่างไปจากวินโดวส์เอ็นที ถ้าหากคุณตั้งใจจะรันแอพพลิเคชัน Win32 บนแพลตฟอร์มทั้งสองล่ะก็ เห็นทีเราต้องมาทำความเข้าใจเรื่องความแตกต่างเหล่านี้ไว้ให้จงดี มิฉะนั้นจะเสียเวลาในการปรับปรุงการพัฒนาและการแก้บักในบางขั้นบางตอน

เราจะดูข้อจำกัดภายใต้ชิคาโกพร้อมไปกับการเปรียบเทียบกับความสามารถของวินโดวส์เอ็นที คุณจะได้เห็นสิ่งที่เรียกว่า "ข้อจำกัด" ได้ชัดเจนขึ้นกว่าพูดถึงอาการ "ไร้ความสามารถ" ของชิคาโกอย่างเดียว

เรื่องทั่วไป

ฟังก์ชันและเมสเสจ Win32 บางตัว เช่น ที่เกี่ยวกับสวัสดิภาพ (Security) และการล็อกเข้าเหตุการณ์ (Event Logging) ไม่ถูกสนับสนุนจากชิคาโก ชิคาโกจะมีแค่รูทีนที่จะทำให้แอพพลิเคชันซึ่งดีไซน์สำหรับโอเอสอื่นที่สนับสนุน Win32 รับบนชิคาโกได้โดยปราศจากข้อผิดพลาดเท่านั้น

การบริหารวินโดวส์

มีการบริการวินโดวส์อันเป็นฟีเจอร์ 16 บิตอยู่บ้าง ซึ่งนี่ก็เป็นข้อบังคับสำหรับพารามิเตอร์ในบางฟังก์ชันและเมสเสจ และยังจำกัดการจัดเก็บภายในอีกด้วย ตัวอย่างเช่น ปริมาณเท็กซ์ในอีดิตคอนโทรลมาตรฐานจะถูกจำกัดอยู่ในช่วง 64 เค (ถ้าเท็กซ์ไฟล์ของคุณใหญ่และต้องการเขมือบหน่วยความจำมากกว่านี้ก็หมดสิทธิ์) ในบางกรณีชิคาโกก็มีฟีเจอร์ใหม่ให้ใช้เพื่อหลีกเลี่ยงข้อบังคับและข้อจำกัดเหล่านี้ อย่างเช่นริชเท็กซ์ (rich text) อีดิตคอนโทรลซึ่งจะทำให้ปริมาณเท็กซ์มากได้เท่าที่ทรัพยากรในมือที่ใช้ได้จริงทั้งหมดจะเอื้ออำนวย

ชิคาโกยอมรับวินโดวส์แฮนเดิ้ลและเมนูแฮนเดิ้ลได้อย่างละ 16,364 ตัว ฟังดูเป็นหมื่นนี่ยังถือว่าน้อยนะครับเมื่อเทียบกับวินโดวส์เอ็นที แต่อย่างน้อยมันกีรับได้มากกว่าบนวินโดวส์ 3.1 แล้วกัน

พารามิเตอร์ wParam สำหรับฟังก์ชัน SendMessageCallback, SendMessageTimeout และ SendNotifyMessage จะถูกจำกัดที่หนึ่งค่า 16 บิต

ภายใต้ชิคาโก พารามิเตอร์ wParam ในเมสเสจของลิสต์บ็อกซ์ เช่น LB_INSERTSTRING หรือ LB_SETITEMDATA จะถูกจำกัดที่หนึ่งค่า 16 บิตเช่นกัน ผลกระทบที่เกิดขึ้นก็คือลิสต์บ็อกซ์จะไม่สามารถรับรายการได้เกิน 32,768 ชิ้น

แต่ถึงแม้จำนวนรายการจะถูกจำกัด แต่ขนาดโดยรวมของรายการในลิสต์บ็อกซ์ก็ไม่ถูกจำกัดตามคุณมีหน่วยความจำแค่ไหนมันก็ใหญ่ได้แค่นั้น นี่เป็นข้อแตกต่างที่ดีกว่าขนาดข้อมูลที่ใหญ่ได้ไม่เกิน 64 เค ในวินโดวส์ 3.1

ภายใต้ชิคาโก ฟังก์ชันเกี่ยวกับคีย์บอร์ดเลเอาต์ เช่น Activate KeyboardLayout, GetKeyboardLayoutName, LoadKeyboardLayout และ UnloadKeyboardLayout สามารถโหลดได้เพียงหนึ่งคีย์บอร์ดในหน่วยความจำแต่ละขณะ ซึ่งถ้าหากอยู่ภายใต้วินโดวส์เอ็นทีแล้ว คุณสามารถโหลดได้หลายคีย์บอร์ด (ซึ่งมาขึ้นทะเบียนอยู่ในหน่วยความจำ) และสามารถใช้ได้ในลักษณะของการสลับกัน แต่ถ้าคุณจะทำอย่างนั้นภายใต้ชิคาโก คุณจะต้องโหลดและรีโหลดทีละตัว

ภายใต้ชิคาโก มีเพียงเดสก์ท็อปเดียวที่ถูกใช้ขณะระบบกำลังรัน ถึงแม้จะมีฟังก์ชันอย่าง GetThreadDesktop และ SetThreadDesktop ไว้ก็ไร้ประโยชน์

แอพพลิเคชันเมสเสจที่เป็นไพรเวทจะต้องถูกนิยามเหนือ WM_USER + 0 × 100 ซึ่งนี่จะทำให้แน่ใจได้ว่าจะไม่มีการชนกันโครมคราระหว่างไพรเวทเมสเสจกับเมสเสจของไดอะล็อกคอนโทรล

GDI (Graphics Device Interface)

ชิคาโกใช้ความสามารถระดับ 16 บิตในการกำหนดตำแหน่งร่วมเอ็กซ์วายสำหรับเท็กซ์และกราฟิกในช่วง ±32K แต่วินโดวส์เอ็นทีจะใช้โลกใหม่ที่ได้มาจากความสามารถระดับ 32 บิตในการกำหนดตำแหน่งในช่วง ±2G ถ้าหากคุณผ่านตำแหน่งร่วม 32 บิตสู่ฟังก์ชัน-

เท็กซ์และกราฟิกภายใต้ชิคาโกระบบจะตัด 16 บิตส่วนบนของตำแหน่งร่วมออกก่อนจะดำเนินการปฏิบัติการตามคำสั่ง

ภายใต้ชิคาโก รีเจียน (region) จะถูกวางจาก 32-bit ฮีป (heap) ฉะนั้นจึงอาจใหญ่ได้พอ ๆ กับหน่วยความจำที่เหลือใช้ได้ (อันนี้ในวินโดวส์ 3.1 จำกัดแค่ 64 เค) อย่างไรก็ดี ลอจิคอลออปเจ็กต์อื่น ๆ ทั้งหมดจะปันส่วนกันภายในโลคอลฮีป 64 เค และจำนวนแฮนเดิ้ลของรีเจียนไม่สามารถมีได้เกิน 16 เค

ออปเจ็กต์กายภาพจะปรากฏอยู่ทั่วหน่วยความจำ ดังนั้นจึงไม่ถูกจำกัดไบต์แต่อย่างใด

ภายใต้ชิคาโก ปากกากับแปรงจะมีข้อจำกัดหลายต่อหลายอย่าง ฟังก์ชัน ExtCreatePen สนับสนุนเพียงสีทึบ (Solid Color - PS_SOLID) เท่านั้น และสไตล์ของปากกาเหล่านี้จะถูกสนับสนุนใน paths เท่านั้น

```
PS_ENDCAP_FLAT
PS_ENDCAP_ROUND
PS_ENDCAP_SQUARE
PS_JOIN_BEVEL
PS_JOIN_MITER
PS_JOIN_ROUND
```

ชิคาโกไม่สนับสนุนเครื่องหมายและประจุดในสไตล์ของปากกา เป็นต้นว่า PS_DASH หรือ PS_DOT ในเส้นกว้าง ๆ สไตล์ของแปลงชื่อ BS_DIBPATTERN จะถูกจำกัดอยู่ในวง 8 x 8

ชิคาโกไม่สนับสนุนแฟลก CBM_CREATEDIB สำหรับฟังก์ชัน CreateDIBitmap มันใช้ CreateDIBSection เพื่อสร้าง DIB (Device-Independent Bitmap) แทน โดยที่ฟังก์ชัน CreateDIBSection จะมีอยู่ในรีลีสหน้าของวินโดวส์เอ็นที

สำหรับ DIB ขนาด 16 หรือ 32 บิตต่อพิกเซลนั้น biCompression ซึ่งเป็นสมาชิกในโครงสร้าง BITMAPINFOHEADER จะถูกเซตเป็นศูนย์ (คืออยู่ในสถานะไม่ถูกอัดแน่น) แทน BI_BITFIELDS ถ้าหากสีของมาสก์ได้ดังนี้

```
16bpp มาสก์สีฟ้าคือ 0x001F, เขียวคือ 0x03E0, และแดงคือ 0x7C00 บิตสูง ๆ จะไม่ถูกใช้ 32bpp มาสก์สีฟ้าคือ 0x000000FF, เขียวคือ 0x0000FF00, และแดงคือ 0x00FF0000 ไบต์สูง ๆ จะไม่ถูกใช้
```

ในกรณีดังกล่าว มาสก์ของ DWORD สามสีจะไม่ถูกใช้ แต่รีลีสหน้าของวินโดวส์เอ็นทีจะเข้ากันได้กับฟอร์แมตดังกล่าว ชิคาโกไม่สนับสนุน biCompression เท่ากับ biBitFields ในทุกไดรเวอร์

การวาดออปเจ็กต์ในการลบจะต่างกันเล็กน้อยภายใต้ชิคาโกกับวินโดวส์เอ็นที นั่นคือภายใต้วินโดวส์เอ็นที หากคุณพยายามที่จะลบออปเจ็กต์ที่ถูกวาดโดยปากกาหรือแปรงในขณะที่ยังถูกเลือกเข้าไปอยู่ใน DC ฟังก์ชัน DeleteObject จะไม่ทำงาน ส่วนภายใต้ชิคาโก ฟังก์ชันดังกล่าวจะทำงาน แต่ผลลัพธ์ออกมาเป็นออปเจ็กต์ที่จะถูกทำลายไปเมื่อ DC ถูกลบทิ้ง

เมื่อสร้าง path ขึ้นมาภายใต้ชิคาโก จะมีแต่ฟังก์ชัน MoveTo, LineTo, PolyBezier และ PolyBezierTo เท่านั้นที่ถูกบันทึก

ภายใต้ชิคาโก ฟังก์ชัน GetGraphicsMode และ SetGraphicsMode สนับสนุนแฟลก GM_COMPATIBLE เท่านั้น ส่วนแฟลก GM_ADVANCED จะไม่ถูกสนับสนุน

ถึงแม้ชิคาโกจะไม่มีข้อบังคับใดเกี่ยวกับฟังก์ชัน PlayEnhMetaFile และ PlayEnhMetaFileRecord ไฟล์กับเรคอร์ดที่ฟังก์ชันเหล่านี้ทำงานจะเข้าข่ายข้อจำกัดดังที่กล่าวถึงไปแล้ว อย่างเช่นฟังก์ชันจะเพิกเรคอร์ดที่พยายามวาดเกินตำแหน่งร่วมขนาด 16 บิต

บริการจากระบบ (System Services - Kernal)

ข้อผิดพลาดของโค้ดส่วนเกิน (Extended Error Code) ที่รีเทอร์นจากฟังก์ชัน GetLastError อาจจะเหมือนหรือไม่เหมือนภายใต้ชิคาโกและวินโดวส์เอ็นที ความแตกต่างนี้ประยุกต์กับข้อผิดพลาดของโค้ด ส่วนเกินที่ถูกผลิตโดยการเรียก GDI, การบริหารวินโดวส์, และฟังก์ชันบริการจากระบบ

ชิคาโกไม่สนับสนุนไฟล์ไอโอประเภท asynchronous ยกเว้นในอุปกรณ์ที่เป็นซีเรียล (serial) ดังนั้นฟังก์ชัน ReadFile และ WriteFile จะไม่ทำงานหากคุณนำรีเจียนที่เหลื่อมกันเข้าไปในอุปกรณ์อะไรก็ตามที่ไม่ใช่ซีเรียล

ฟังก์ชัน GetOverlappedResult จะทำงานก็ต่อเมื่อเป็นอุปกรณ์ซีเรียลหรือบนไฟล์ที่ถูกเปิดใช้โดยฟังก์ชัน DeviceIoContol

ปัจจุบันฟังก์ชัน GetSystemTime จะรีเทิร์นเวลาที่เป็นโลคอล ไม่ใช่ UTC แต่ในอนาคตฟังก์ชันนี้จะรีเทอร์น UTC

เรื่องความเที่ยงตรงของเวลาบนไฟล์บนระบบ FAT ไฟล์นั้น อาจมีความผิดพลาดได้สองวินาที ถ้าหาก

ชิคาโกติดต่อผ่านเน็ตเวอร์กสู่ระบบไฟล์ที่แตกต่างกัน ความเที่ยงตรงของเวลาจะถูกจำกัดโดยอุปกรณ์ที่ติดต่อด้วยตัวนั้น

ภายใต้วินโดวส์เอ็นที ฟังก์ชัน DeleteFile จะไม่ทำงานหากคุณพยายามที่จะลบไฟล์ซึ่งถูกเปิดใช้สำหรับไอโอธรรมดาหรือเป็น memory mapped file ภายใต้ชิคาโก ฟังก์ชัน DeleteFile สามารถลบไฟล์ดังกล่าวได้ คุณต้องระวังให้ดี อย่าลืมปิดไฟล์ก่อนลบด้วยฟังก์ชัน มิฉะนั้นอาจเกิดการสูญข้อมูลและทำให้แอพพลิเคชันล่มเอาได้ง่าย ๆ

ภายใต้ชิคาโก การใช้ฟังก์ชัน MapViewOfFileEx เพื่อสร้าง shared memory mapped files ปรากฏในแอดเดรสที่เหมือนกันตลอดระบบประมวลผลระดับ 32 บิต หากคุณผ่านพารามิเตอร์ lpvBase ของฟังก์ชัน MapViewOfFileEx แล้วมันเวอร์กก็เป็นการประกันได้ว่าหน่วยความจำรีเจียนเดียวกันใช้ได้ในทุกการประมวลผล แต่นี่ไม่เป็นอย่างนั้นภายใต้วินโดวส์เอ็นที คือ MapViewOfFileEx จะไม่ทำงานกับการประมวลผลใด ๆ ที่ได้รับหน่วยความจำรีเจียนไว้ใช้แล้ว

ภายใต้ชิคาโก ไฟล์แฮนเดิ้ลที่ถูกใช้เพื่อสร้างออปเจ็กต์ที่เป็น File-Mapping ต้องไม่ถูกใช้ในการเรียกฟังก์ชันไฟล์ไอโอครั้งต่อ ๆ มา อย่างเช่น ReadFile กับ WriteFile เป็นต้น โดยทั่วไปแล้ว ถ้าหากไฟล์แฮนเดิ้ลหนึ่ง ๆ ถูกใช้ในการเรียกฟังก์ชัน CreateFileMapping สำเร็จ คุณก็อย่าไปใช้แฮนเดิ้ลนั้นเว้นแต่ว่าจะปิดออปเจ็กต์ที่เป็น File-mapping เสียก่อน

ภายใต้ชิคาโก ฟังก์ชัน Flush-InstructionCache จะรีเทิร์นค่า true เสมอ เนื่องจากชิคาโกสนับสนุนโปรเซสเซอร์เดี่ยวเท่านั้น

ข้อจำกัดของระบบ Win32s

Win32s ให้ฟังก์ชัน Win32 และเมสเสจบางตัวต่างจากวินโดวส์เอ็นที หากคุณตั้งใจจะรันแอพพลิเคชันของ Win32 บนทั้งสองแพลตฟอร์ม คุณต้องเข้าใจข้อแตกต่างซึ่งจะช่วยให้งานพัฒนากับการแก้บักเหลือน้อยที่สุด

การบริหารวินโดวส์

Win32s จะมอบหมายให้วินโดวส์ 3.1 จัดเตรียมไดอะล็อกคอนโทรลมาตรฐาน เช่น ลิสต์บ็อกซ์, คอมโบบ็อกซ์, และอีดิตคอนโทรล โดยที่ Win32s เป็นผู้แปลเมสเสจระหว่างคอนโทรลกับแอพพลิเคชัน Win32

คอนโทรลมีขนาดจำกัด (บนวินโดวส์เอ็นทีไม่มี) เป็นต้นว่าอีดิตคอนโทรลตัวหนึ่งจะถูกจำกัดอยู่ในช่วงหน่วยความจำไม่เกิน 64 เค สำหรับเท็กซ์, ลิสต์บ็อกซ์จะเก็บข้อมูลได้เพียงหนึ่งฮีปขนาด 64 เค เช่นกัน ด้วยเหตุฉะนี้ แอพพลิเคชัน Win32 จึงสามารถสร้างไฟล์ใหญ่ ๆ ด้วยอีดิตคอนโทรลในวินโดวส์เอ็นที และไม่สามารถอ่านไฟล์เดียวกันในอีดิตคอนโทรลตัวเดียวกันเมื่อแอพพลิเคชัน Win32 รันบนวินโดวส์ 3.1

คอนโทรลสนับสนุนข้อจำกัดเฉพาะบนจำนวนข้อมูลที่มันจะรับ เช่น EM_SETLIMIT สำหรับอีดิตคอนโทรล ทางออกอย่างหนึ่งคือกำหนด lowest-common-denominator สำหรับคอนโทรลเพื่อให้แน่ใจว่ามันสามารถทดแทนกันได้เมื่อข้ามเพลตฟอร์ม ทางออกอื่น ๆ แล้วแต่ข้อกำหนดในแต่ละแอพพลิเคชัน

Win32s ไม่สนับสนุนเมสเสจ EM_SETHANDLE และ EM_GETHANDLE เมสเสจเหล่านี้อนุญาตให้ใช้แฮนเดิ้ลของหน่วยความจำอันเป็นโลคอลระหว่างแอพพลิเคชันกับระบบร่วมกันได้ ใน Win32s นั้น โลคอลฮีปของแอพพลิเคชันมีขนาด 32 บิต และถูกกำหนดมาจากโกลบอลฮีป ฉะนั้นวินโดวส์ 3.1 จึงไม่สามารถแปลแฮนเดิ้ลที่เป็นโลคอลได้

เท็กซ์สำหรับดีดิตคอนโทรลหลาย ๆ บรรทัดจะถูกเก็บไว้ในโลคอลฮีปขนาด 16 บิต ซึ่งก็หมายความว่าปริมาณเท็กซ์ในอีดิตคอนโทรลถูกจำกัดอยู่ในช่วงไม่เกิน 64 เค และเพื่อที่จะอ่าน/เขียนอีดิตคอนโทรลหลาย ๆ บรรทัด แอพพลิเคชันควรจะได้ใช้เมสเสจ WM_GETTEXT และ WM_SETTEXT

เมสเสจของแอพพลิเคชันที่เป็นไพรเวทจะต้องถูกนิยามเหนือ WM_USER + 0×100 ที่ทำอย่างนี้ก็เพื่อให้แน่ใจว่าจะไม่มีการชนกันระหว่างไพรเวทเมสเสจกับเมสเสจของไดอะล็อกคอนโทรลบนวินโดวส์ 3.1

เมื่อเรียกใช้ฟังก์ชัน PeekMessage แอพพลิเคชัน Win32 ไม่ควรกรองเมสเสจที่มีเพื่อ private window classes ของวินโดวส์ 3.1 ใด ๆ (เช่น ปุ่มหรือสโครลบาร์) เมสเสจสำหรับคอนโทรลเหล่านี้จะถูกแม็พกับค่าที่ต่าง

กันใน Win32 และการเช็คความสำคัญของ mapping ก็ใช้เวลามากด้วย

identifier ของวินโดวส์ตัวลูก (Child Window) บน Win32s จะต้องเป็นค่า sign-extended ขนาด 16 บิต ซึ่งนี่จะกันการใช้ค่าพอยน์เตอร์ขนาด 32 บิตออกจากความเป็น IDs ของวินโดวส์ตัวลูก

ขนาดของเมสเสจคิวในวินโดวส์เอ็นทีจะเพิ่มขึ้นเรื่อย ๆ ส่วนวินโดวส์ 3.1 จะมีขนาดเมสเสจคิวอันเป็นดีฟอลต์คือ 8 ซึ่งสามารถเปลี่ยนไปโดยการเรียก SetMessageQueue การเรียกนี้จะถูกสนับสนุนใน Win32s และเป็น NOP บนวินโดวส์เอ็นที

SetClipboardData จะต้องถูกเรียกด้วยโกลบอลแฮนเดิ้ล ถ้าไม่อย่างนั้นแล้ว ข้อมูลจะไม่สามารถถูกเข้าถึงโดยแอพพลิเคชันอื่น

แอพพลิเคชันเป็นผู้กำหนดค่ารีเทอร์นสำหรับฟังก์ชันไดอะล็อกบ็อกซ์ โดยวิธีการผ่านพารามิเตอร์ nResult ไปยังฟังก์ชัน EndDialog พารามิเตอร์ตัวนี้มีชนิดข้อมูลเป็น int ซึ่งขนาดก็ 32 บิตสำหรับแอพพลิเคชัน Win32

TranslateMessage จะสนับสนุนงานบนวินโดวส์เอ็นทีมากกว่าบนวินโดวส์ 3.1 Win32s จัดเตรียมภารกิจของวินโดวส์ 3.1 ซึ่งจะรีเทอร์นค่า true เมื่อคีย์หรือตัวเลขใด ๆ ถูกพิมพ์ (ซึ่งก็คือการเกิดของลำดับ KEYDOWN/KEYUP นั่นเอง) บนวินโดวส์เอ็นทีนั้น ฟังก์ชันและคีย์ลูกศรจะให้ค่ารีเทิร์นของ TranslateMessage เป็น ture เช่นกัน

IDs ของทรัพยากร (identifier ใน resource file) จะต้องมีค่าเป็น 16 บิต

ขนาดของตารางทรัพยากรจะถูกจำกัดอยู่ที่ 32 เค (ไม่รวมข้อมูลทรัพยากรนะครับ)

GDI (Graphics Device Interface

วินโดวส์ 3.1 จำกัดจำนวนดีไวซ์คอนเท็กซ์ของแคชแค่ 5 การเรียก GetDC จะได้มาซึ่ง DC ของแคช ฉะนั้นมันเป็นสิ่งจำเป็นที่จะเรียก ReleaseDC ก่อนที่จะเช็คเมสเสจคิว มิฉะนั้นแอพพลิเคชันอื่นในระบบอาจเจอพิกัดจำนวน DC ของแคชน้อยกว่าเดิมหนึ่ง ส่วนวินโดวส์เอ็นทีนั้นจะไม่มีจำนวนจำกัด DC ของแคช อย่างไรก็ตามแอพพลิเคชันไม่ควรไปใช้สิ้นเปลืองนัก

เป็นสิ่งจำเป็นที่จะตามแบบจำลอง GetDC/ReleaseDC เพื่อแทนกันได้ทั้งสองแพลตฟอร์มและการบริหารหน่วยความจำที่ดี

วินโดวส์ 3.1 อนุญาตการลบออปเจ็กต์ซึ่งถูกวาดด้วยปากกาและแปรงในขณะที่พวกมันยังอยู่ใน DC อย่างไรก็ตาม หน่วยความจำที่ถูกจัดสรรเพื่อออปเจ็กต์ดังกล่าวจะยังคงไม่ถูกปลดปล่อยไปไหนจนกว่าการประมวลผลจะสิ้นสุด ส่วนวินโดวส์เอ็นทีจะไม่รับการเรียกใช้ DeleteObject แอพพลิเคชันไม่ควรลบออปเจ็กต์ที่อยู่ใน DC

SetDIBitsToDevice ไม่สนับสนุนบิตแม็พในหน่วยความจำ DCs และจะถูกจำกัดหน่วยความจำไม่เกิน 2 เมกะบิตเมื่อจะใช้แสดงบิตแม็พ

จะมีกระบวนการ cleanup เพื่อให้แน่ใจว่าออปเจ็กต์ที่ถูกจัดสรร โดยแอพพลิเคชันที่จบการทำงานจะถูกทำลายและหน่วยความจำจะถูกล้างเพื่อใช้งานอื่น กระบวนการ cleanup บนวินโดวส์ 3.1 มีคุณภาพดีกว่าบนวินโดวส์ 3.0 แต่ก็ไม่สมบูรณ์แบบเท่ากับบนวินโดวส์เอ็นที เนื่องจากแอพพลิเคชัน Win32 ที่รันบนวินโดวส์ 3.1 จะต้องรอคอยเป็นเวลาเนิ่นนานด้วยข้อจำกัดของพลังระดับ 16 บิต

อันเนื่องมาจากวินโดวส์ 3.1 อยู่ในโลกของระบบจุดร่วม (Coordinate) ระดับ 16 บิต ส่วนวินโดวส์เอ็นทีอยู่ในระดับ 32 บิต ฉะนั้นจึงเป็นเรื่องจำเป็นที่จะต้องใช้ข้อจำกัดของวินโดวส์ 3.1 ในการพัฒนาแอพพลิเคชัน Win32 ที่เป็นกราฟิกซึ่งคุณคิดจะให้มันรันบนทั้งสองแพลตฟอร์ม

วินโดวส์เอ็นทีอนุญาตให้แอพพลิเคชัน Win32 เรียก SetCursor และระบุแฮนเดิ้ลของไอคอนวินโดวส์เอ็นทีสนับสนุนสีของไอคอนและสีของเคอร์เซอร์ ส่วนวินโดวส์ 3.1 สนับสนุนแค่สีไอคอนและเคอร์เซอร์แบบโมโนโครม (พูดง่าย ๆ ว่าไม่หลากสีนั่นแหละครับ) ฉะนั้นคุณจึงไม่สามารผ่านแฮนเดิ้ลของไอคอนไปยัง SetCursor บน Win32s

บริการจากระบบ

การประมวลผลของ Win32s ที่ถูกสตาร์ทโดยแอพพลิเคชันวินโดวส์ 3.1 ควรจะออกมาจาก WinExec เนื่องมาจาก LoadModule ของวินโดวส์ 3.1 จะไม่สามารถสตาร์ตการประมวลผลของ Win32s ถ้าหากคุณใช้ LoadModule ของ Win32 ล่ะก็ จะพบความแตกต่างใน IpCmdLine ในโครงสร้าง LOADPARAMS32 ซึ่งถูกผ่านไปยังฟังก์ชันต่อไปนี้

- บนวินโดวส์เอ็นที IpCmdLine

ชี้ไปที่ Pascal-style สตริงตัวหนึ่งที่มีคอมมานด์ไลน์ที่ถูกฟอร์มขึ้นอย่างถูกต้อง ไบต์แรกของสตริงมีจำนวนไบต์ในสตริง ที่เหลือเป็นอาร์กิวเมนต์ของคอมมานด์ไลน์ (ไม่นับชื่อของการประมวลผลลูก (Child Process)) ถ้าหากไม่มีอาร์กิวเมนต์ของคอมมานด์ไลน์ พารามิเตอร์ตัวนี้จะต้องชี้ไปที่สตริงความยาวศูนย์ (คือไม่มีอักขระอะไรเลย แต่จะเป็น NULL ไม่ได้)

● บน Win32 lpCmdLine ชี้ไปที่สตริงที่ยังใช้อยู่ (Null-Teminated String) ซึ่งมีคอมมานด์ไลน์ที่ถูกฟอร์มขึ้นอย่างถูกต้อง และสตริงจะต้องมีความยาวไม่เกิน 120 ไบต์ด้วย

VirtualAlloc จัดเตรียมฟีเจอร์บนวินโดวส์เอ็นทีซึ่งอนุญาตการประมวลผลให้ขอใช้ที่หน่วยความจำในแอดเดรสเสมือน (Virtual Address) ได้ อันนี้ถูกสนับสนุนบน Win32s ครับ แต่แอพพลิเคชันไม่ควรไปติดอยู่กับช่วงแอดเดรสที่ขอได้มากนัก วินโดวส์เอ็นทีจัดสรรหน่วยความจำในแอดเดรสที่ว่างระดับต่ำให้ 2GB ส่วน Win32s จัดสรรที่ว่างระดับสูงให้ 2GB เท่ากัน ทั้งนี้แอพพลิเคชันสามารถขอดูแอดเดรสว่างโดยใช้ GetSystemInfo

อย่าไประบุ GMEM_FIXED ตอนใช้ GlobalAlloc อันนี้จะได้ผลลัพธ์คือ locked pages ในหน่วยความจำบน Win32s ซึ่งทำให้การจัดสรรจะถูกจำกัดอยู่กับหน่วยความจำทางกายภาพแทนที่จะเป็นข้อจำกัดของหน่วยความจำเสมือน (ที่เรียกกันว่า pageable memory allocation ไงครับ)

การอิมพลิเมนต์ Sleep ของ Win32 API ต่างจาก Win32s และวินโดวส์เอ็นที เราจะเรียกฟังก์ชันนี้บน Win32S ว่า "Yield" ซึ่งจะรีเทอร์นทันทีถ้าหาก

1. ไม่มีแอพพลิเคชันอื่นใดมีเมสเสจในคิวรอการประมวลผล

2. แอพพลิเคชันที่ทำการเรียก Yield ยังคงมีเมสเสจในคิวของมัน

เป็นไปไม่ได้ครับที่จะบล็อกการประมวลผลเพื่อหน่วงเวลาตามต้องการเหมือนอย่างในวินโดวส์เอ็นที เพราะว่าวินโดวส์ 3.1 นั้นเป็นการทำงานหลายอย่างแบบ Cooperative (คือมีการคุยกันตลอดเวลาว่าใครเป็นใคร ทำอะไรอยู่ ร่วมงานแบบถ้อยทีถ้อยอาศัยกัน) ไม่ใช่ทำงานหลายอย่างแบบ Preemptive (เล่นแบบเผด็จการครับ ไม่ต้องรอ ฉันจะเอาก่อนเสียอย่าง) การหน่วงเวลาควรทำโดยใช้ลูป PeekMessage ซึ่งเรียก GetSystemTme เพื่อควบคุมเวลาตามความต้องการ

คุณไม่สามารถเรียก DeleteFile บนวินโดวส์เอ็นที เพื่อลบไฟล์ซึ่งถูกเปิดไว้แล้ว (สำหรับไอโอสามัญหรือ memory file) บนเอ็มเอสดอส (ซึ่งก็คือ Win32s ด้วย) คุณสามารถลบได้ ถึงแม้ว่า SHARE.EXE ถูกโหลดขึ้นมาแล้วก็ตาม การลบไฟล์ดังกล่าวอาจให้ผลเป็นการสูญเสียข้อมูลและทำให้แอพพลิเคชันล่มได้ ฉะนั้นคุณควรระมัดระวังให้มาก เมื่อใช้ตัววิบัติอย่าง DeleteFile และต้องแน่ใจว่าไฟล์นั้นไม่ถูกใช้แล้วจริง ๆ

เรื่องความเที่ยงตรงของเวลาอาจคลาดได้สองวินาที ซึ่งอันนี้เป็นข้อจำกัดของเอ็มเอสดอส

GetPrivateProfileSection และ GetProfileSection จะทำงานตอนอินนิทไฟล์ที่มีคีย์เอกลักษณ์ (Unique Keys) เท่านั้น อย่างเช่น

```
[TestSection]
Entry1 = 123
Entry1 = 456
Entry1 = 789
Entry2 = ABC
```

GetPrivateProfileSection จะรีเทอร์นสตริง Entry1 = 123 ออกมาสามตัว (ไม่สนใจการใส่ค่าครั้งต่อ ๆ มา เนื่องจากเป็น Unique Keys จุดสำคัญก็คือคุณต้องรู้ว่ามันเอาเฉพาะค่าแรกที่ใส่ไป)

วินโดวส์ 3.1 สนับสนุนค่าเดี่ยว (สตริง) ต่อหนึ่งคีย์ในทะเบียนฐานข้อมูลของมัน ส่วนวินโดวส์เอ็นทีจะสนับสนุนหลายค่าต่อหนึ่งคีย์ และมี APIs ใหม่เพื่อดำเนินการกับฐานข้อมูล

แอพพลิเคชัน Win32 และ DLLs ควรจะถูกลิงค์กับหน่วยความจำเรียงติดกัน 4 เค อันนี้เป็นดีฟอลต์สำหรับลิงเกอร์ LINK32 ซึ่งมาพร้อมกับ Win32 SDK (Software Development Kit) อันที่จริงหน่วยความจำเรียงติดกัน 64 เค ก็ถูกสนับสนุนด้วย แต่ 4 เค จะทำให้การบริหารความจำมีประสิทธิภาพดีกว่าเยอะในการแชร์แอดเดรสระดับ 2GB บนวินโดวส์ 3.1

Networking

WinSocket API สนับสนุนการเรียกทั้งชนิด blocking และ nonblocking สเปกของ WinSocket บ่งว่าการเรียกชนิด blocking ควรหลีกเลี่ยงที่จะทำบนระบบอย่างวินโดวส์ 3.1 ซึ่งไม่ใช่การทำงานร่วมกันชนิด Preemptive ฉะนั้นแอพพลิเคชันที่ใช้ Win32 WinSocket ควรใช้การเรียกชนิด nonblocking ซึ่งจะทำให้รันได้อย่างราบรื่นทั้งบน Win32s และวินโดวส์เอ็นที

วินโดวส์เอ็นทีสนับสนุนเหล่าฟีเจอร์ NetBIOS ซึ่งพวกมันไม่ถูกสนับสนุนบน Win32s อันเนื่องมาจากการแชร์หน่วยความจำแบบ nonpreemptive ซึ่งเป็นการออกแบบเพื่อวินโดวส์ 3.1

ฟิลด์ ncb_event ของโครงสร้าง NCB จะถูกเพิกเนื่องจาก Win32s ไม่อิมพลิเมนต์ event ของวินโดวส์เอ็นที และวินโดวส์เอ็นทีก็รักษาชื่อเทเบิ้ลไว้ชื่อเดียวขณะที่ Win32s รักษาชื่อเทเบิ้ลได้ทีเดียวทั้งระบบ

มัลติมีเดีย

Win32s สนับสนุน APIs บนวินโดวส์เอ็นทีที่เกี่ยวกับเสียงทั้งหมดยกเว้น MIDI callbacks ฉะนั้นแฟล็ก CALL_FUNCTION ของ midiOutOpen กับ midiInOpen จึงไม่ถูกสนับสนุนทั้งสองตัว และสำหรับ timeSetEvent กับ timeKillEvent ก็ไม่ถูกสนับสนุนเช่นกัน

บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน)

ผู้ผลิตวารสาร และ หนังสือชั้นนำของประเทศ อาทิ วารสารไมโครคอมพิวเตอร์ วารสารเซมิคอนดักเตอร์อิเล็กทรอนิกส์, วารสาร UP-DATE, วารสาร BEST BUY, วารสาร HOW TO in business และหนังสือตำราวิชาการทุกสาขาวิชา ต้องการบุคลากรมาร่วมงานในตำแหน่งต่อไปนี้

พนักงานขายโฆษณา

คุณสมบัติ วุฒิการศึกษาระดับปริญญาตรีทุกสาขา ทั้งชายและหญิงที่มีใจรักงานขายโฆษณา ไม่จำเป็นต้องมีประสบการณ์

สนใจสอบถามรายละเอียด เขียนจดหมายสมัครงานหรือ สมัครได้ด้วยตนเองที่

แผนกบุคคล

บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน) 800/43-45 ซอยตระกูลสุข ถนนอโศก-ดินแดง เขตดินแดง กรุงเทพฯ 10400 โทรศัพท์ 2486280 (อัตโนมัติ 20 สาย) โทรสาร 2465999